The Mach-Zehnder Interferometer
A Smart Remote Experiment Based on a Software Template

Wissam Halimi, Christophe Salzmann, and Denis Gillet
Institute of Electrical Engineering
Swiss Federal Institute of Technology Lausanne (EPFL), Station 9, 1015 Lausanne, Switzerland
Emails: {wissam.halimi, christophe.salzmann, denis.gillet}@epfl.ch

***Abstract*—Hands-on laboratory sessions are one of the pillars of science and engineering education. It is reputed that institutions find it difficult to provide enough laboratory setups that cover all taught scientific topics due to logistical and budgetary limitations. With the recent advances in web technologies enabling real-time interaction, remotely accessing physical devices became possible, and so the rise of remote experimentation as a solution to the mentioned limitations. The development and deployment of remote laboratories is still a tedious process for lab providers, given the need to write new applications for each new lab. In this paper, we propose a software template for lab providers that will alleviate some of their concerns by following the Smart Device Paradigm and Specifications and abiding to software engineering rules to produce a reusable and flexible solution. We will show how this in-progress template can be used for implementing a remote Mach-Zehnder Interferometer.**

***Keywords*—remote laboratories; online education; physics; quantum physics; mach-zehnder; optics; service oriented computing**

I. Introduction

Hands-on laboratory sessions are considered to be an essential component of the process of acquiring scientific knowledge [1] [2]. But many educational institutions run short on budget to provide enough setups or any setups at all for their students. Remote experimentation is considered as a temporal and spacial convenience (anytime and anywhere access) for students to do experiments. In addition to increasing the reach of pedagogy and sharing expensive resources, remote experimentation allows students who are geographically dispersed to have access to common resources [3]. A typical example is the availability of a certain experimental setup at one educational institution, which is used once or twice a year, and having enough availability-time to share the resources with other institutions incapable of acquiring such setups or reaching the premises. The Mach-Zehnder Interferometer is an example of one of the experiments that are hard to acquire by schools due to the expensive material. The Mach-Zehnder Interferometer is known for its versatile configuration, and hence its applications are wide. It is used in a number of fundamental topics in quantum mechanics: counterfactual definiteness, quantum computation, quantum cryptography, quantum logic, and Elitzur-Vaidman bomb tester; in addition to many others. For example, in optical telecommunications, it is used as an electro-optic modulator for phase and as an amplitude modulation device for light that transports the information. So possessing such a setup is an asset to support STEM education at school. For instance, quantum physics was added as an optional topic in Swiss high-schools this year. For many students the conceptual assimilation of quantum mechanics can be rather hard, owing to the counter-intuitive nature of quantum phenomena. By providing a tool that makes understanding quantum physics an easier task, the intimidation caused by the difficulty to apprehend quantum phenomena is surpassed, and hence encouraging students to pursue scientific and engineering majors. To support the teaching of the behavior of photons and electrons, we implement a remote MachZehnder interferometer. The interferometer helps the students in perceiving how quantum phenomena deviate from our classical everyday experience [4]. In this paper we present a remote implementation of the Mach-Zehnder Interferometer, configured to study light interference. We show how building a lab application following the Smart Device Paradigm and Specifications makes the remote lab experiment flexible and highly reusable in different ways. We propose a software package, which we claim to be a software template for lab providers to use for the development and deployment of new remote labs in order to speedup the process. The paper is structured as follows: first we detail the Smart Device Paradigm and Specifications. Second we provide a brief explanation of the structure of the Mach-Zehnder Interferometer and the supported experimentation scenarios. Then, we present the ongoing work on the software template and how it is adapted for this experiment. In a last section, we explain how client applications can be customized with the provided framework.

II. Related Work

A. The Smart Device Paradigm

Remotely controlling physical processes is traditionally implemented following a coupled architecture that ties the process server and the client application. Typically the process server senses and controls the remote devices, while the client application provides an interface for users to interact with the remote setup. Adopting such an architectural topology hinders the versatility of usage of physical processes. The

978-1-4673-8246-5/16/$31.00 ©2016 IEEE

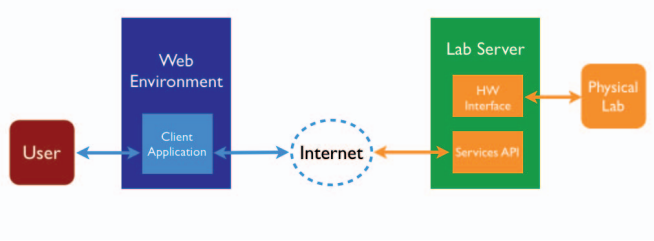

Fig. 1: The Smart Device architecture for remote labs

Smart Device Paradigm is an architectural framework for connecting remote experiments online, which proposes to separate the usually-coupled two components of remotely controlled laboratories: the Client and the Lab Server applications. This is done by equipping the Lab Server side with some intelligence and specifying requirements according to Service Oriented Computing concepts. The paradigm starts from the definition of intelligence for objects. In literature, physical devices are said to be smart or intelligent if they support ICT (Information and Communication Technologies). In other words, if they are able to collect, process, produce information, reason, take action, and communicate with other smart devices. Other requirements for smart objects include having an identity (an IP for example) and being self-aware (memory and status tracking) [5] [6]. When devices have such capacities, they can be remotely controlled [6]. Having a Lab Server side built according to the Smart Device Paradigm renders the complete setup smart and remotely controlled. In order to provide access points for the Client Application to the setup, the Smart Device Specifications provide an API for the Lab Server as described in the next sub-section.

B. The Smart Device Specifications

When built following the Smart Device Paradigm, a remote lab is exposed on the Internet as a set of services through an API. The API describes two categories of services: internal and external. They respectively deal with services destined to guarantee the security of the laboratory and are not accessible by the Client Application, and services to sense/control the setup and are accessible by the Client Application. The components of the lab setup are categorized as sensors or actuators. Typically sensors reflect the current state of the lab, while actuators alter the state to the lab. Sensors and actuators are described with a set of attributes in order to make access to them resemble accessing services. The totality of the lab is represented in a metadata file in the JSON format. This file is served by the Lab Server, and accessible to the Client Application. The Smart Device Specifications stipulate that the communication protocol is WebSocket. This choice was made based on the full-duplex channel communication supported by this protocol, and hence providing a true real-time communication with the remotely controlled lab [7].

III. The Mach-Zehnder Interferometer

As mentioned in an earlier section, the Mach-Zehnder Interferometer is used to teach a variety of topics in physics and other fields. In this paper, we present a configuration of this device meant to study quantum mechanics. The presented arrangement of the Mach-Zehnder experiment demonstrates the light interference phenomena by splitting a monochromatic light beam, and reflecting it repeatedly as shown in Fig. 2.

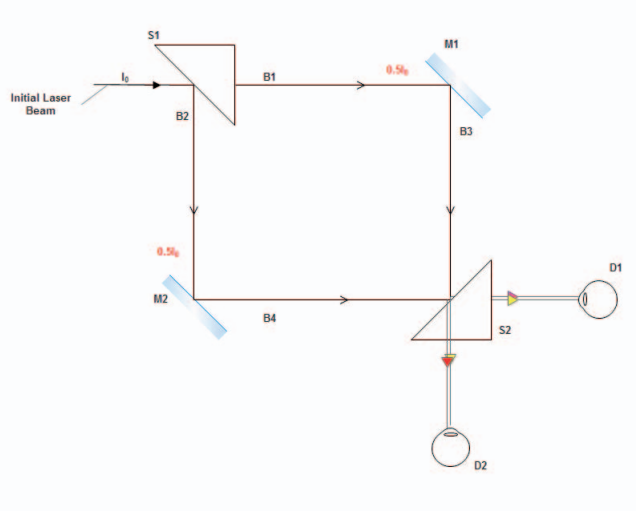

Fig. 2: The Mach-Zehnder interferometer in the classical mode

A. The Classical Mode

In the classical mode of the Mach-Zehnder Interferometer, the concept of light interference is demonstrated using a one-color light beam. The experiment goes as follows:

- A monochromatic light beam is sent out of the laser and first hits the beam splitter S1. S1 causes the light to split into 2 beams with equal intensities of $\frac{I_0}{2}$. I_0 being the intensity of the original beam. B1 is the transmitted beam and B2 is the reflected one.
- The incident beam on mirror M1 gets 100% reflected and takes the path B3. The same goes for the reflected beam B2 which hits mirror M2 and gets fully reflected and take the path B4.
- Both beams B3 and B4 hit the beam splitter S2 and get again divided into 2 paths, one transmitted and the other reflected. The intensities of the resulting beams are equal and are $\frac{I_0}{4}$.
- The resulting beams hit detectors D1 and D2.

The classical interference phenomenon results form the superposition of waves: at both detectors, the amplitudes of the 2 components of the initial laser B1 and B2 superpose, each with an intensity of $\frac{I_0}{4}$. Assuming that the waves travel in a positive direction that we denote by x, the resulting electric field of the laser beam can be mathematically expressed by:

$$E = E_0 \bar{x} \cos(kx - \omega t) \tag{1}$$

Where $\bar{x}$ is the unit vector on the x direction, $k = \frac{2\pi}{\lambda}$ is the wavenumber (λ is the wavelength), and ω is the angular frequency.

$$I = \frac{1}{c\mu_0} \overline{E_0 \bar{x} \cos(kx - \omega t)^2} \tag{2}$$

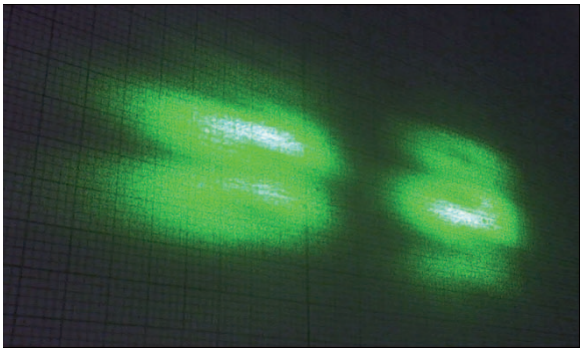

Fig. 3: Light interference patterns on detectors

Where the bar denotes the time average taken over a period $T = \frac{2\pi}{\omega}$. It is possible, therefore, to establish the following relations for the laser beam:

$$I_0 \sim E_0^2$$

$$\frac{I_0}{4} \sim \frac{E_0^2}{4} = \frac{{E_0}^2}{2} \tag{3}$$

For an incidence angle of 45°the phase shift between the transmitted and the reflected light beam components is $\frac{\pi}{2}$, corresponding to a path length difference of $\frac{\lambda}{4}$, where λ is the wavelength. Hence, the beam components that fall on detector D1 undergo two consecutive reflections and one transmission remaining in phase. We observe a constructive interference pattern on D1. This can be mathematically proven as follows:

$$\begin{aligned} E &= \frac{E_0}{2}\bar{x}\cos(kx - \omega t + \pi) + \frac{E_0}{2}\bar{x}\cos(kx - \omega t + \pi) \\ &= -E_0\bar{x}\cos(kx - \omega t) \end{aligned} \tag{4}$$

This means that the intensity of the laser beam shown on D1 is recovered, but with a phase inversion. Looking at the consecutive paths taken by the two components of the light beam arriving at D2, we see that one of the components experiences only one reflection, while the other experiences three consecutive reflections. We can conclude that the beam components that hit D2 are delayed by π, which corresponds to a destructive interference pattern. In summary, the totality of the beam energy is preserved for the detected beam at D1 and energy is lost for the beam arriving at D2. This can be formulated as follows [4]:

$$\begin{aligned} E &= \frac{E_0}{2}\bar{x}\cos(kx - \omega t + \frac{\pi}{2}) + \frac{E_0}{2}\bar{x}\cos(kx - \omega t + \frac{3\pi}{2}) \\ &= 0 \end{aligned} \tag{5}$$

Fig. 3 illustrates the resulting destructive and constructive light interference patterns. Notice respectively that on the left of the figure, the center is dark followed by alternating illuminated and then darkened out areas. While the opposite happens on the right part of the figure.

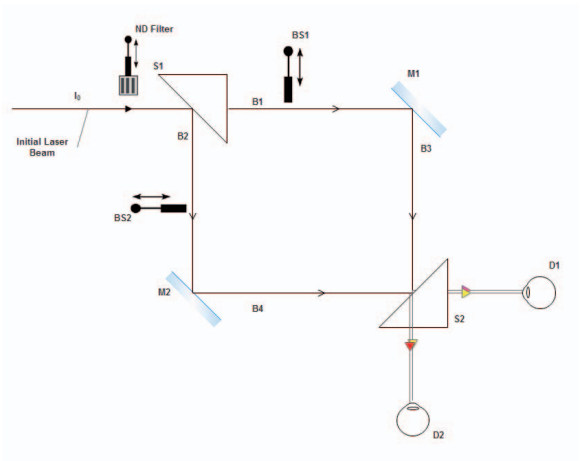

Fig. 4: The Mach-Zehnder interferometer in the quantum mode

B. The Quantum Mode

In the quantum mode, we want to see how single photons behave when performing the same experiment as in the classical mode with a laser beam (continuous jet of photons). In order to do that, it is necessary to drastically decrease the intensity of the light beam. This is done by either one of the following options, or both:

- Introducing the ND (Neutral Density) Filter F1 in the laser path
- Introducing Beam Shutters BS1 and/or BS2 in the laser path

Students will conclude that in the quantum mode, where only one photon is emitted, the same results as in the classical mode will be gradually collected. Fig. 4 illustrates how we can pass from the classical to the quantum mode as explained above.

Fig. 5 shows some elements of the physical lab arranged in a test configuration while adapting the setup for remote access. It shows the laser beam that would go through a half-mirror, then two consecutive mirrors, before hitting a last half-mirror and hitting one the detectors in place. Some parts of this setup are static while others are movable. For example, the second mirror can be moved forward and backward with a Piezo-controller, this helps experimenting with several concepts that lie within interferometry (changing the dimensions of the interference patterns).

IV. A Software Template for the Lab Server

The Smart Device Paradigm simplifies the task of developing and deploying remote laboratories for lab providers by freeing them from the burden of custom-building associated client applications. This also offers them the possibility of broadly utilizing their setups by several institutions by providing an API for accessing the lab services. Allowing the development of various client applications, each suited to the needs of teachers and students without having to modify the Lab Server side [7]. While this leverages a portion of lab owners concerns,

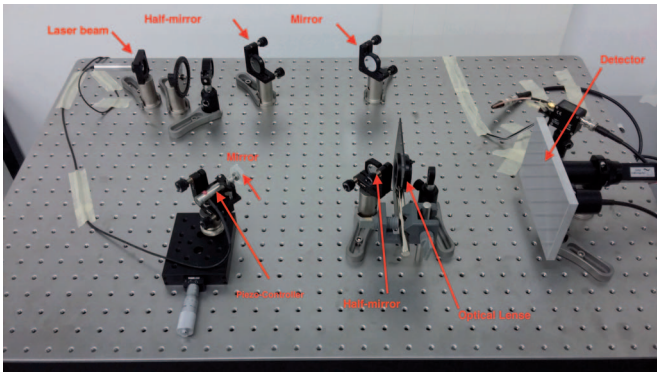

Fig. 5: Preliminary physical setup of the Mach-Zehnder Interferometer

building a remote lab is still a tedious process. This is because for every new lab, new hardware interfacing needs to be done, and the Lab Server application needs to be re-written.

In this section, we propose an in-progress software template for the Lab Server side that will give lab owners a starting point on their work.

In software engineering, software requirements are defined as being the conditions to meet for new software, which take into account the possible needs of different stakeholders. In our case, the stakeholders are the lab owners, and the goal is to allow the development of new remote labs using a general solution. Those requirements are guaranteed by software specifications, which describe how the software package provides a promise to implement the requirements [8]. In the following we detail our definition of the *Template Requirements*, and the *Template Specifications*.

1) Template Requirements: We expect our software package to be:

a. *Scalable*: Scalability means it is rather an easy task to expand the software to make it handle more work.
b. *Readable*: Readability is translated into the easiness to inspect the code and structure of the software application, in addition to understanding its functionality.
c. *Maintainable*: Maintainability is the ability to easily add new features to the software without affecting the original implemented features [9].

2) Template Specifications: We will promise the fulfilment of the *Template Requirements* by implementing the following specifications:

a. *Data Specifications*: This category of the specifications designates the data formats used in the exchange of requests and responses. The template assumes that all incoming requests and all outgoing responses are in the JSON format. The template produces and consumes JSON encoded data.
b. *Behavioral Specifications*: Under this title, we define the components that make up the software application and how they communicate:

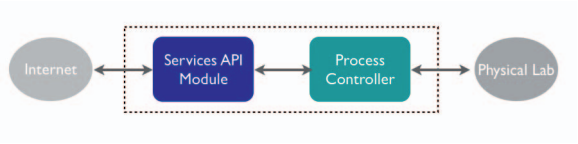

Fig. 6: Software template high-level architecture

- The template is made out of two main modules: the Services API Module and the Process Controller.
- The Services API Module handles incoming requests from the Client Application, parses them adequately to the convenience of the Process Controller and forwards the requests to it. It also receives the data to send back to the Client Application from the Process Controller and properly formats the response as per the API definition.
- The Process Controller encompasses the logic for controlling the lab setup. Its main functionality is to receive commands and associated data from the Services API Module ready to be consumed by the services. It invokes the controls on the lab setup, and gathers readings from the sensors. The affected and collected data is then forwarded to the Services API Module for response packaging.
- We call the data interchanged between the Services API Module and the Process Controller "Interaction Data". While we name the data exchanged between the Physical Lab and the Process Controller "Experimental Data".

Fig. 6 illustrates the above: the Process Controller encompasses a hardware-interfacing module which translates the Experimental Data into formats understandable by the physical lab or Process Controller.

A. Software Template Structure

The software template presented in this paper is written with LabVIEW, and deployed on the myRIO board. It is possible to deploy the template on a different system supporting LabVIEW, while adapting the hardware interfacing code.

The template implements the following mechanisms:

a. Listening to requests and building responses
b. Parsing requests and dispatching them
c. Connecting the parsing mechanism to the hardware interfacing module

B. How to Use the Software Template

As mentioned, the software template sets the skeleton of the application implementing the Lab Server side of the Smart Device Architecture, and follows the specifications presented in section IV. The provided software package can be used by the lab owner as follows:

1. Adapt the *metadata* file to adequately describe the services of the considered laboratory

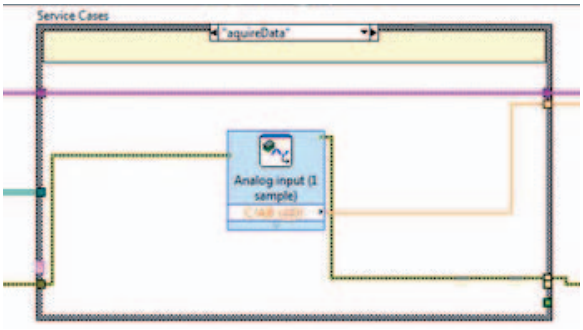

Fig. 7: Hardware interfacing case example

2. Implement the hardware interfacing respective to the equipment to the lab in question

In section V we detail the structure of the metadata file, and how it is used by the Client Application and the Lab Server in sending and processing requests.

Fig. 7 shows an example of hardware interfacing when the lab is supposed to acquire an analog signal connected to pin AI0 of connector C.

Accordingly, the lab owner creates more structure cases and handles their hardware. The template implements 4 services to provide a basic setup of communication chains between the different components of the application making it easy to augment. The implemented services are: turn ON and OFF process, and two dummy services that respectively read and write from and to 2 different pins of the myRIO board, and the service that provides the *metadata* file of the lab encompassing the API. The progress on the software template can be followed on Github: https://github.com/react-epfl/rl_templates/tree/master/LabVIEW. The repository is public and the code is available for download.

V. The Remote Mach-Zehnder Experiment

The experiment that we are presenting in this paper is a work-in-progress for bringing a remote version of the Mach- Zhender Interferometer online as per the Smart Device Architecture and Specifications, and by using the software package presented in the previous section.

Following the template, the lab server is composed of two modules: the Services API Module which provides access points to client applications, and the Process Controller which takes care of sensing and controlling the physical lab. In this section, we will explain how the lab is described as a set of services through an API served by the Services API Module. As mentioned before, the Process Controller is ad-hoc to an experiment and is hooked to the Services API Module in the template, two subjects unnecessary to detail for the purpose of this paper.

The Smart Device Paradigm specifies a systematic way of describing the API in a JSON file that we call the "*metadata*" file. It is an adapted version of the Swagger API[1], where an extension is added in order to provide support for the WebSocket protocol. The main parts of it are:

1. *The metadata service parameters*: They are 4 parameters: the *apiVersion* which identifies the version of the API this lab is using, *swaggerVersion* tells which Swagger version this API structure is based on, *basePath* refers to the url to be used in order to access the lab, and *info* which provides a high-level description of the lab (title, description, contact, license, and license url):

```
"apiVersion": "1.0.0",
"swaggerVersion": "1.2",
"basePath": "http://remotemachzehnder.com",
"info": {
  "title": "The Mach-Zehnder
  Interferometer",
  "description": "Remote experimentation
    for light interference in classical and
    quantum modes",
  "contact": "wissam.halimi@epfl.ch",
  "license": "Apache 2.0",
  "licenseUrl": "http://www.apache.org/
    licenses/LICENSE-2.0.html" }
```

2. *The authorisation schema*: This part tells how the lab server authenticates and authorizes users to have access on the physical lab. Schemas can include token exchanges, user sign in, etc...

```
"authorizations": {
  "login": {
    "type": "username",
    "passAs": "loginAuth",
    "keyName": "password"
    }}
```

3. *The concurrency schema*: This describes whether the lab is synchronous (real-time control), or asynchronous (batch), in addition to describing how multiple users asking for access are handled at the same time.

```
"concurrency": {
  "interactionMode": "synchronous",
  "concurrencyScheme": "roles",
  "roleSelectionMechanism": ["race",
    "fixed role"],
  "roles": [ {
    "role": "controller",
    "selectionMechanism": ["race"] }
    ]}
```

4. *The APIs*: This part of the metadata file details all the services that lets the user client interact with the physical lab. It tells how the interfacing application should send requests to sense or control a sensor or actuator respectively. It also provides information on

[1] http://swagger.io/

how is formatted the response, in order to allow for an easy parsing mechanism. An *api* element has *protocol*, *produces*, and *operations* fields. *protocol* identifies the communication protocol adopted for servicing this API element. *produces* specifies the data format of the response. *operations* describes the service as an object type that is described in the section models.

```
"apis": [
{
  "protocol": "websocket",
  "produces": [
    "application/json"
    ],
  "operations": [{...}]
  }]
```

5. *The models*: Just like in object oriented programming, each object has class, but in this case, a model. The models section provides a formal description of the objects used in the apis section in order to describe the structure of requests and responses.

```
"ClientResponse": {
  "id": "ClientResponse",
    "properties": {
    "method": {"type": "string"},
    "clients": {
      "type": "array",
      "items": {
        "\$ref": "Client"
        }}
    }
}
```

A. Client Application and Lab Server Communication

Knowing the host address of the remote lab, the metadata file can be retrieved with an HTTP request to the host-name/metadata according to the Smart Device Specifications. For this lab for example, the link is: http://shindig2.epfl.ch/remote_machzehnder_metadata.html
Since the metadata file is in JSON format, and is structured as per the presented schema in this section, the file can be parsed or read, and hence getting all necessary information for accessing the sensors and actuators of the laboratory.

B. Example

Suppose that a teacher desires that students can only turn ON and OFF the laser beam, control the ND Filter, one beam shutter, and see the video feed of the experiment. In this case, the client application asks access for those services in dedicated requests. The steps for each request would be:

- Get metadata for the desired service
- Format the request according and send it
- Parse the response and embed it in the client application as desired

Notice that the example client application does not use all the sensors and actuators available in the lab. It only uses a subset of the equipment, and hence customizing the user client without imposing modifications on the lab server side is possible. The client application and the lab server are decoupled, which allows for the complete customization of the user client, not just in terms of experimentation scenarios, but also in terms of language support, scientific annotations or terminologies.

VI. Conclusion and Future Works

In this paper, we propose a new way of building remote labs with the use of software templates. An approach that we believe is time-saving for lab providers, who wish to decrease their efforts for developing and deploying labs, and increase the visibility of their setups. This paper presents the ongoing efforts for building the software template, and validating it as we go with a concrete example: the remote Mach-Zehnder Interferometer. We started by presenting the planned experimental scenarios, then presented the framework in which the template is conceptualized and implemented. We later explained how the template is adapted in order to implement the remote experiment. In the last section, we explain how client applications can be automatically built using the provided API. This work-in-progress still requires work to reach completion, namely the next steps are finalizing the template, finishing the implementation of the lab server side, and bringing the experiment online.

Acknowledgment

This work is partially funded by the Go-Lab project as part of the FP7 EU Framework. We would like to thank Stephane Chamot and Philippe Lobello of the Gymnase de Morges in Switzerland, for their involvement and commitment in making this work a success and providing the future opportunity to conduct a case study with the students using the lab hosted at their institution.

References

[1] D. Lowe, "Mools: Massive open online laboratories: An analysis of scale and feasibility," in *Remote Engineering and Virtual Instrumentation (REV), 2014 11th International Conference on*. IEEE, 2014, pp. 1–6.

[2] B. Dasarathy, K. Sullivan, D. C. Schmidt, D. H. Fisher, and A. Porter, "The past, present, and future of moocs and their relevance to software engineering," in *Proceedings of the on Future of Software Engineering*. ACM, 2014, pp. 212–224.

[3] J. Ma and J. V. Nickerson, "Hands-on, simulated, and remote laboratories: A comparative literature review," *ACM Computing Surveys (CSUR)*, vol. 38, no. 3, p. 7, 2006.

[4] A. Pereira, F. Ostermann, and C. Cavalcanti, "On the use of a virtual mach–zehnder interferometer in the teaching of quantum mechanics," *Physics Education*, vol. 44, no. 3, p. 281, 2009.

[5] C. Gutierrez, J. Garbajosa, J. Diaz, and A. Yague, "Providing a consensus definition for the term" smart product"," in *Engineering of Computer Based Systems (ECBS), 2013 20th IEEE International Conference and Workshops on the*. IEEE, 2013, pp. 203–211.

[6] C. Thompson, "Smart devices and soft controllers," *Internet Computing, IEEE*, vol. 9, no. 1, pp. 82–85, 2005.

[7] C. Salzmann, S. Govaerts, W. Halimi, and D. Gillet, "The smart device specification for remote labs," in *Remote Engineering and Virtual Instrumentation (REV), 2015 12th International Conference on*. IEEE, 2015, pp. 199–208.

[8] I. Sommerville and G. Kotonya, *Requirements engineering: processes and techniques*. John Wiley & Sons, Inc., 1998.

[9] N. Instruments, *LabVIEW Core 3 Course Manual*, 2015.